Turbo Code Decoding
Applied to Trellis-Coded Modulation

Bruce E. Wahlen and Calvin Y. Mai
Space and Naval Warfare Systems Center
San Diego, CA 92152

October 5, 1998

Contents

1 Introduction

Turbo codes were first introduced in a 1993 paper by Berrou, Glavieux, and Thitimajshima [1] describing their new coding method. Their simulation results of a rate-$\frac{1}{2}$ turbo code together with BPSK modulation in an additive white Gaussian noise (AWGN) channel showed coding gains of up to 11 dB, which is within 0.5 dB of the Shannon capacity limit. Their method is a generalization of concatenated codes, a technique developed by Forney [2] to achieve significant coding gain at reasonable complexity. The concatenation of Reed-Solomon (RS) block codes and convolutional codes, for example, has been used quite effectively for many years in a variety of applications, including audio compact discs, deep-space telemetry, and in commercial and military modems.

Originally, turbo codes were defined in [1] as the *parallel* concatenation of *two* constituent *convolutional* codes, separated by a *single* pseudorandom interleaver, and employing an *iterative* decoding algorithm (Fig. 1). Now, the original concept has been extended to include *serial* concatenation, *more than two* constituent codes (CC's), *block* CC's, and *multiple* interleavers. Results of simulations performed at the Jet Propulsion Laboratory (JPL), cited in [3], demonstrate that turbo codes can achieve significant coding gains with orders of magnitude reductions in complexity relative to some powerful RS/convolutional concatenated codes over the approximately AWGN deep-space channel. In the deep-space applications of JPL delay can be virtually ignored since the data are processed off-line; however, in many other applications, such as those involving voice communication, delay cannot be ignored. In general, the decision to use turbo codes involves an analysis of the trade-offs between the *benefits* of very large coding gain achieved at reasonable complexity and the *costs* of increased delay, related to interleaver length and number of decoding iterations, and bandwidth expansion.

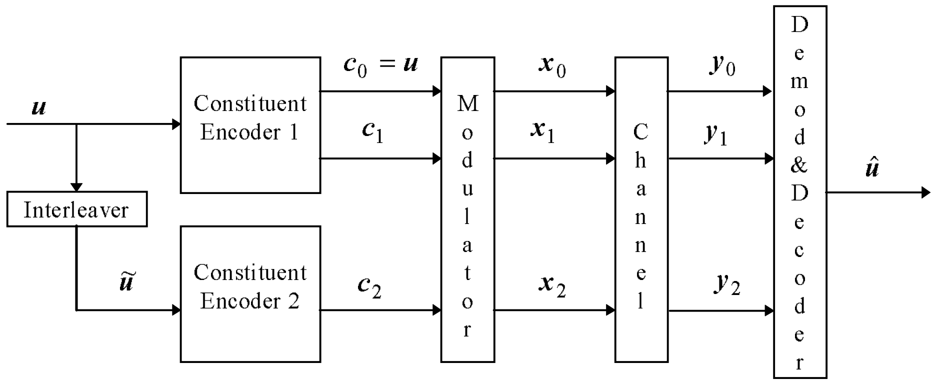

Figure 1: Parallel concatenated turbo code.

As the near-capacity gains claimed for turbo codes have been confirmed and widely reported in the literature, the range of applications of turbo codes has expanded to many areas of communications. This paper presents the results of applying turbo codes to Trellis-Coded Modulation (TCM); that is, we replace the standard convolutional codes used in TCM with more powerful turbo codes, in an attempt to achieve the large coding gains at reasonable complexity reported for turbo codes, while maintaining the high bandwidth efficiency associated with TCM methods. For the TCM part of the combination, we use a variation of Ungerboeck's technique [4] known as Pragmatic TCM [5].

The remainder of our paper is organized into two main sections: a tutorial description of turbo code decoding in general (section 2), with simulation results for a rate-$\frac{1}{3}$ turbo code with BPSK modulation and for a rate-$\frac{1}{2}$ turbo code with QPSK modulation; and a description of our turbo-coded Pragmatic TCM (section 3), with simulation results for a rate-$\frac{2}{3}$ turbo code with 8-PSK modulation and a discussion of delay.

2 Turbo Code Decoding

The Viterbi algorithm [6] is the most widely used method to decode convolutional codes, preferred over the Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm [7] in most applications. The Viterbi algorithm, based on the maximum likelihood (ML) decision criterion, minimizes the error probability in detecting the whole sequence (block, message, or word); while the BCJR algorithm, based on the maximum *a posteriori* (MAP) decision criterion, minimizes the error probability in detecting symbols or bits [7–9]. Even though the Viterbi algorithm is not optimal in terms of bit error probability, in most applications its bit error performance nearly matches that of the optimal but more complex BCJR algorithm. For this reason the Viterbi algorithm has been preferred over the BCJR algorithm for the decoding of convolutional codes.

For turbo codes, however, the size of the state-space is too large to perform Viterbi decoding [8], so Berrou *et al.* [1] developed a suboptimal iterative decoding procedure requiring *a posteriori* symbol probabilities which the MAP-based BCJR algorithm can produce. Hence, the BCJR algorithm has become an integral part of the iterative decoder for turbo codes. Simulation results in [1] and in many other references demonstrate near-Shannon limit performance of the iterative decoder and suggest that it often converges to the optimal decoding solution. The exact convergence properties remain an open research question; see [10–12], for example.

2.1 The BCJR Algorithm

While Bahl *et al.* [7] derived this algorithm to solve the general problem of estimating the *a posteriori* probability (APP) of the states and transitions of a Markov source observed through a noisy discrete memoryless channel, they also showed that the algorithm could be applied to linear block and convolutional codes as a special case. We describe the BCJR algorithm in the context of convolutional codes, following the development of Benedetto, Montorsi, Divsalar, and Pollara [9].

2.1.1 Assumptions and Notation

We assume the same communication system (Fig. 2) as in [9] in which a source produces a discrete-time indexed sequence of information symbols, $\boldsymbol{u} = (u_1, \ldots, u_k, \ldots, u_n)$, where each u_k belongs to the information symbol alphabet, U, containing I symbols. An encoder then maps the sequence of information symbols into a sequence of code words, $\boldsymbol{c} = (c_1, \ldots, c_k, \ldots, c_n)$, where each $c_k \in \mathcal{C}$. The code, $\mathcal{C}$, is a subset of the K-fold Cartesian product of the code symbol alphabet, C, containing M symbols, that is, $c_k \in \mathcal{C} \subseteq C^K$. See below (Example 1) for a binary symbol alphabet example.

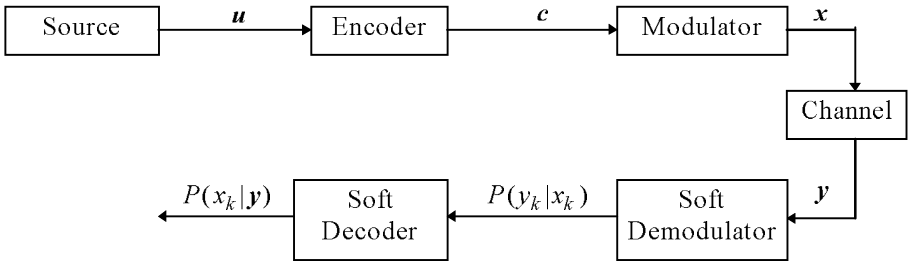

Figure 2: Communication system.

Example 1 *Let* $U = \mathbb{F}_2 \triangleq \{0, 1\}$*, the finite field containing two elements, so that* $\boldsymbol{u}$ *is a sequence of binary symbols, or bits. Consider the rate-*$\frac{1}{2}$ *binary recursive systematic convolutional encoder with feedback connection vector* $\boldsymbol{g_0} = [111]$ *and feedforward connection vector* $\boldsymbol{g_1} = [101]$*, depicted in shift register form below (Fig. 3). This encoder generates code words,* $c_k = (c_{k1}, c_{k2}) \in \mathcal{C}$*, for which* $\mathcal{C} = C^K$*,* $C = \mathbb{F}_2$ *and* $K = 2$ *in the above notation, and where the code words are generated by the following system of equations:*

$$\begin{aligned} c_{k1} &= u_k, \\ a_k &= u_k + a_{k-1} + a_{k-2}, \\ c_{k2} &= a_k + a_{k-2}. \end{aligned}$$

After the encoding operation (Fig. 2) a modulator performs a one-to-one map of the sequence of code words into a sequence of channel input symbols, $\boldsymbol{x} = (x_1, \ldots, x_k, \ldots, x_n)$, where each x_k belongs to a set X containing M symbols.[1] The channel input symbols, x_k, are then transmitted through a stationary, memoryless channel, producing a sequence of received observations,

[1]For simplicity we have assumed that the cardinality of the modulator alphabet, X, is equal to the cardinality of the code symbol alphabet, C.

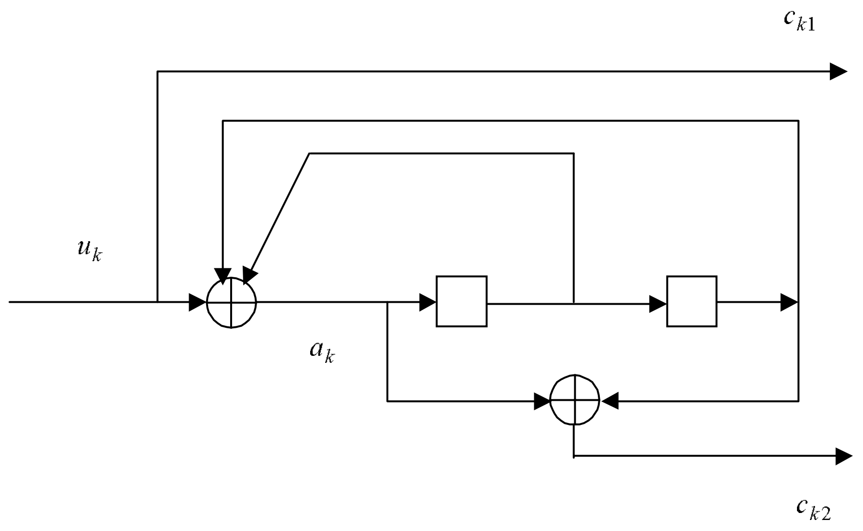

Figure 3: Rate-$\frac{1}{2}$ binary recursive systematic convolutional encoder of Example 1.

$\boldsymbol{y} = (y_1, \ldots, y_k, \ldots, y_n)$. The received observations are input to a soft-output demodulator, producing a sequence of distributions, $P(y_k \mid x_k)$, conditioned on the channel symbols. Finally, a soft-output decoder, produces a sequence of probability distributions, $P(x_k \mid \boldsymbol{y})$.

We assume further that the encoder is a time-invariant recursive systematic convolutional (RSC) encoder with N states, $S = \{S_1, \ldots, S_N\}$, and an associated trellis representation. With these assumptions the code sequences $\boldsymbol{c}$ and the corresponding transmitted signal sequences $\boldsymbol{x}$ can be identified with paths in the trellis, and also $\boldsymbol{c}$ and $\boldsymbol{x}$ can be uniquely associated with a time-indexed state sequence $\boldsymbol{s} = (s_0, \ldots, s_k, \ldots, s_n)$, where $s_k \in S$ and where the first and last states, s_0 and s_n, are assumed to be known by the decoder. As in [9] we introduce the following notation relative to sections of the trellis immediately before and after the kth time instant:

1. $S_i^-(u')$ is the unique *precursor*[2] of S_i defined by the information bit u' determining the transition $S_i^-(u') \to S_i$.

2. $S_i^+(u)$ is the unique *successor*[2] of S_i defined by the information bit u determining the transition $S_i \to S_i^+(u)$.

3. To each transition in the trellis a signal x and code word c are associated, which depend on the state from which the transition originated and on

[2]For the class of recursive convolutional codes, the precursor (successor) of S_i is uniquely determined by the information bit u' (u).

the information bit u determining that transition. This dependence will be denoted by $x(u', S_i)$ or $c(u', S_i)$ when the transition *terminates* at S_i and by $x(S_i, u)$ or $c(S_i, u)$ when the transition *originates* from S_i.

A pictorial description of this notation, like that provided in [9], is included below (Fig. 4), followed by an explanation of the notation (Example 2) in terms of the RSC encoder of Example 1.

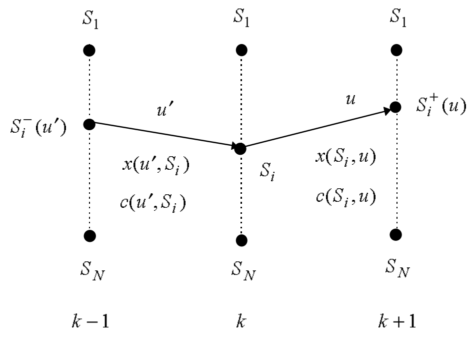

Figure 4: Representative trellis section.

Example 2 *Given the rate-$\frac{1}{2}$ binary RSC encoder of Example 1 (Fig. 3) and all possible combinations of input information symbol, u_k, and current state of the shift register, $(a_{k-1}a_{k-2})$, we have derived the shift register input, a_k, the next state, $(a_k a_{k-1})$, and the output code word, (c_{k1}, c_{k2}), displayed in the table below. Letting $s_k = S_i = (01)$, for example, it is seen from the table that $S_i^-(0) = (11)$, $c(0, S_i) = (0,1)$, $S_i^+(1) = (00)$, and $c(S_i, 1) = (1,1)$; while the values of $x(0, S_i)$ and $x(S_i, 1)$ are undetermined since the modulation scheme is unspecified.*

Input Information Symbol	Current State	Shift Register Input	Next State	Output Code Word
0	(00)	0	(00)	(0,0)
1	(00)	1	(10)	(1,1)
0	(01)	1	(10)	(0,0)
1	(01)	0	(00)	(1,1)
0	(10)	1	(11)	(0,1)
1	(10)	0	(01)	(1,0)
0	(11)	0	(01)	(0,1)
1	(11)	1	(11)	(1,0)

Table 1: States of the RSC encoder of Example 1.

2.1.2 Computation of the APP's

The original form of the BCJR algorithm applied to this communication system, calculates the APP's of encoder states and transitions, namely

$$\sigma_k(S_i, u) \triangleq P(u_k = u, s_{k-1} = S_i \,|\, \boldsymbol{y}), \tag{1}$$

the *a posteriori* transition probabilities at time k, given the *complete* received sequence.[3] We will now show that $\sigma_k(S_i, u)$ is the product of four terms.

First, we partition $\boldsymbol{y}$ into *past* observations, $\boldsymbol{y_1^{k-1}} = (y_1, \dots, y_{k-1})$, the *present* observation, y_k, and *future* observations, $\boldsymbol{y_{k+1}^n} = (y_{k+1}, \dots, y_n)$, and apply Bayes' rule to expand $\sigma_k(S_i, u)$ as

$$\begin{aligned}\sigma_k(S_i, u) = \frac{1}{P(\boldsymbol{y})} \sum_{S_j \in S} & P(\boldsymbol{y_{k+1}^n} \,|\, u_k = u, s_{k-1} = S_i, s_k = S_j, \boldsymbol{y_1^{k-1}}, y_k) \\ & \times P(\, u_k = u, s_{k-1} = S_i, s_k = S_j, \boldsymbol{y_1^{k-1}}, y_k).\end{aligned}$$

After omitting conditioning variables in the first term of the summation, which are irrelevant by the Markov behavior of convolutional codes, we get that

$$\sigma_k(S_i, u) = \frac{1}{P(\boldsymbol{y})} \sum_{S_j \in S} P(\boldsymbol{y_{k+1}^n} \,|\, s_k = S_j) P(u_k = u, s_{k-1} = S_i, s_k = S_j, \boldsymbol{y_1^{k-1}}, y_k). \tag{2}$$

Next, we expand the second term inside the summation in (2) into the product of four terms as

$$\begin{aligned} P(\, u_k = u, s_{k-1} = S_i, s_k = S_j, \boldsymbol{y_1^{k-1}}, y_k) &= P(u_k = u, y_k \,|\, s_{k-1} = S_i, \boldsymbol{y_1^{k-1}}) \\ &\times P(s_k = S_j \,|\, u_k = u, s_{k-1} = S_i, \boldsymbol{y_1^{k-1}}, y_k) \\ &\times P(s_{k-1} = S_i \,|\, \boldsymbol{y_1^{k-1}}) P(\boldsymbol{y_1^{k-1}}). \end{aligned} \tag{3}$$

The first term of the product on the right hand side (RHS) of (3) may be rewritten as

$$\begin{aligned} P(u_k = u, y_k \,|\, s_{k-1} = S_i, \boldsymbol{y_1^{k-1}}) &= P(u_k = u, y_k \,|\, s_{k-1} = S_i) \\ &= P(x_k = x(S_i, u), y_k). \end{aligned} \tag{4}$$

Then, substitution of (4) and of the definition of

$$\alpha_k(S_i) \triangleq P(s_k = S_i \,|\, \boldsymbol{y_1^k}) \tag{5}$$

for the first and third terms, respectively, on the RHS of (3) gives

$$\begin{aligned} P(\, u_k = u, s_{k-1} = S_i, s_k = S_j, \boldsymbol{y_1^{k-1}}, y_k) &= P(s_k = S_j \,|\, u_k = u, s_{k-1} = S_i) \\ &\times P(x_k = x(S_i, u), y_k) \alpha_{k-1}(S_i) P(\boldsymbol{y_1^{k-1}}), \end{aligned} \tag{6}$$

[3]In some expositions of the BCJR algorithm, $\sigma_k(S_i, u)$ is defined equivalently as the *joint* probability of u_k, s_{k-1}, and $\boldsymbol{y}$ rather than the *conditional* probability given here.

where we have dropped irrelevant conditioning variables in the second term on the RHS of (3).

Finally, substituting (6) and the definition of

$$\beta_k(S_i) \triangleq P(\boldsymbol{y}_{k+1}^n \,|\, s_k = S_i) \tag{7}$$

into (2), we get that

$$\sigma_k(S_i,u) = \sum_{S_j \in S} \beta_k(S_j) P(s_k = S_j \,|\, u_k = u, s_{k-1} = S_i) P(x_k = x(S_i,u), y_k) \\ \times \alpha_{k-1}(S_i) P(\boldsymbol{y}_1^{k-1}) / P(\boldsymbol{y}).$$

But, by the uniqueness of the successor of state S_i (section 2.1.1), the second term in this summation equals 1 if $S_j = S_i^+(u)$, and 0 otherwise; therefore, it follows that this sum reduces to the single term

$$\sigma_k(S_i,u) = h_{\sigma_k} \alpha_{k-1}(S_i) \gamma_k(x(S_i,u)) \beta_k(S_i^+(u)), \tag{8}$$

where

$$h_{\sigma_k} \triangleq \frac{1}{P(\boldsymbol{y}_k^n)}$$

is a normalizing constant assuring that $\sum_{S_i,u} \sigma_k(S_i,u) = 1$ and

$$\begin{aligned} \gamma_k(x(S_i,u)) &\triangleq P(x_k = x(S_i,u), y_k) \\ &= P(y_k \,|\, x_k = x(S_i,u)) P(x_k = x(S_i,u)). \end{aligned} \tag{9}$$

From the second line of (9) we observe that $\gamma_k(x(S_i,u))$ is computed from the output of the soft demodulator (Fig. 2) and from knowledge of the *a priori* probabilities of the channel symbols, and further, that the logarithms of the first factor on the RHS are just the *branch metrics* of the Viterbi algorithm. Computation of $\alpha_{k-1}(S_i)$ and $\beta_k(S_i^+(u))$ in (8) is accomplished by means of a *forward* and *backward* recursion, respectively, through the trellis as described in the following section.

2.1.3 Computation of α and β

Applying Bayes' rule to the definition in (5), it is seen that

$$\alpha_k(S_i) = \frac{1}{P(\boldsymbol{y}_1^k)} \sum_{u \in U} \sum_{S_j \in S} P(s_k = S_i \,|\, u_k = u, s_{k-1} = S_j, \boldsymbol{y}_1^k) \\ \times P(u_k = u, s_{k-1} = S_j, \boldsymbol{y}_1^k).$$

After omitting the irrelevant conditioning vector, $\boldsymbol{y}_1^k$, and by the uniqueness of the precursor of state S_i (section 2.1.1), the first term in this sum is simply

$$\begin{aligned} P(s_k = S_i \,|\, u_k = u, s_{k-1} = S_j, \boldsymbol{y}_1^k) &= P(s_k = S_i \,|\, u_k = u, s_{k-1} = S_j) \\ &= \begin{cases} 1, \text{ if } S_j = S_i^-(u) \\ 0, \text{ otherwise.} \end{cases} \end{aligned}$$

Hence, this double summation expression for $\alpha_k(S_i)$ reduces to the single summation

$$\alpha_k(S_i) = \frac{1}{P(\boldsymbol{y_1^k})} \sum_{u \in U} P(u_k = u, s_{k-1} = S_i^-(u), \boldsymbol{y_1^k}). \tag{10}$$

Partitioning $\boldsymbol{y_1^k}$ into the past and present observations, $\boldsymbol{y_1^{k-1}}$ and y_k, respectively, as in section 2.1.2, we see that

$$P(u_k = u, s_{k-1} = S_i^-(u), \boldsymbol{y_1^k}) = P(u_k = u, y_k \,|\, s_{k-1} = S_i^-(u), \boldsymbol{y_1^{k-1}}) \\ \times P(\, s_{k-1} = S_i^-(u) \,|\, \boldsymbol{y_1^{k-1}}) P(\boldsymbol{y_1^{k-1}}).$$

Omitting the irrelevant conditioning term, $\boldsymbol{y_1^{k-1}}$, we recognize that the first term on the RHS of this last equation is $\gamma_k(x(u, S_i))$; while the second term is $\alpha_{k-1}(S_i^-(u))$. Finally, substituting into (10) we see that $\alpha_k(S_i)$ is computed by means of the *forward* recursion

$$\alpha_k(S_i) = h_{\alpha_k} \sum_{u \in U} \alpha_{k-1}(S_i^-(u)) \gamma_k(x(u, S_i)), \tag{11}$$

where

$$h_{\alpha_k} \triangleq \frac{1}{P(y_k)}$$

is a normalizing constant assuring that $\sum_{S_i} \alpha_k(S_i) = 1$ and where the recursion is initialized as

$$\alpha_0(S_i) = \begin{cases} 1, \text{ if } S_i = s_0 \\ 0, \text{ otherwise.} \end{cases} \tag{12}$$

It should be noted that this forward recursion in (11) is, in effect, a weighted sum of the branch metrics of the Viterbi algorithm. See below (Fig. 5) for a pictorial representation of this recursion obtained from [3].

By a similar derivation applied to the definition in (7), it can be shown that $\beta_k(S_i)$ may be computed by the *backward* recursion

$$\beta_k(S_i) = h_{\beta_k} \sum_{u \in U} \beta_{k+1}(S_i^+(u)) \gamma_{k+1}(x(S_i, u)), \tag{13}$$

where h_{β_k} is a normalizing constant assuring that $\sum_{S_i} \beta_k(S_i) = 1$ and where the recursion is initialized as

$$\beta_n(S_i) = \begin{cases} 1, \text{ if } S_i = s_n \\ 0, \text{ otherwise.} \end{cases} \tag{14}$$

Like the $\alpha_k(S_i)$ term the $\beta_k(S_i)$ term is computed, in effect, as a weighted sum of the branch metrics of the Viterbi algorithm. See below (Fig. 5) for a pictorial representation of this recursion.

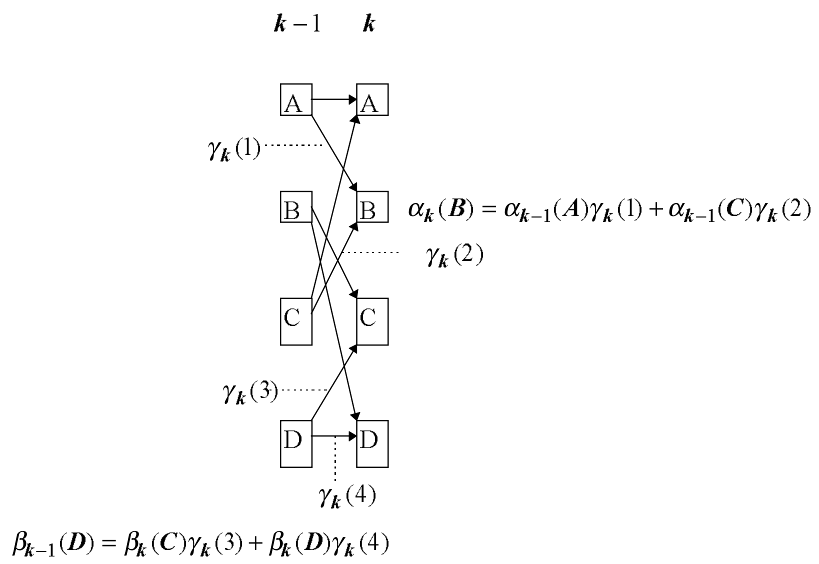
$k-1$
k
A
A
B
B
C
C
D
D
$\gamma_k(1)$
$\gamma_k(2)$
$\gamma_k(3)$
$\gamma_k(4)$
$\alpha_k(B) = \alpha_{k-1}(A)\gamma_k(1) + \alpha_{k-1}(C)\gamma_k(2)$
$\beta_{k-1}(D) = \beta_k(C)\gamma_k(3) + \beta_k(D)\gamma_k(4)$

Figure 5: Forward and backward recursions of the BCJR algorithm.

2.1.4 Computation of γ

The exact form of $\gamma_k(x(S_i,u))$, which is required to calculate $\sigma_k(S_i,u)$, $\alpha_k(S_i)$, and $\beta_k(S_i)$, varies depending on the specific encoder and modulator. To provide a sample calculation, we consider a specific case of the communication system described above (Fig. 2) for which the encoder is a rate-$\frac{1}{2}$ turbo code and the modulator performs a QPSK mapping (Fig. 6). As shown in Figure 11, the rate-$\frac{1}{2}$ turbo code is obtained by puncturing parity bits output from a rate-$\frac{1}{3}$ turbo code, comprising two identical RSC encoders which are concatenated in parallel and separated by an interleaver, denoted as I in the figure. Following the notation in section 2.1.1, the information bits will be denoted by $u_k \in U = \mathbb{F}_2$; the code words will be denoted by $c_k = (c_{k1}, c_{k2}) \in C^2$, where $C = \mathbb{F}_2$; the channel input symbols will be denoted by

$$x_k = (x_{k1}, x_{k2}) \in X = \{(+1,+1),(-1,+1),(-1,-1),(+1,-1)\},$$

defined by the mapping

$$x_{ki} = 2c_{ki} - 1, \text{ for } i = 1,2; \tag{15}$$

and the received observations will be denoted by $y_k = (y_{k1}, y_{k2})$. To emphasize the dependence of the code words on the input bit $u_k = u$ and the encoder state $s_{k-1} = S_i$, we will use the notation

$$\begin{aligned} c_k(S_i,u) &= (c_{k1}(S_i,u), c_{k2}(S_i,u)) \\ &= (u, c_{k2}(S_i,u)), \end{aligned} \tag{16}$$

since $c_{k1} = u_k$ (Fig. 6). Similarly, to denote the dependence of the channel symbols on the input bit and encoder state, we will use the notation

$$\begin{aligned} x_k(S_i,u) &= (x_{k1}(S_i,u), x_{k2}(S_i,u)) \\ &= (2u-1, 2c_{k2}(S_i,u)-1), \end{aligned}$$

after substitution of (16) into (15). It is assumed that the received observations y_k are determined by

$$y_{k1} = x_{k1} + i_k \tag{17a}$$

$$y_{k2} = x_{k2} + q_k, \tag{17b}$$

where i_k and q_k are independent, zero-mean Gaussian random variables with common variance σ^2.

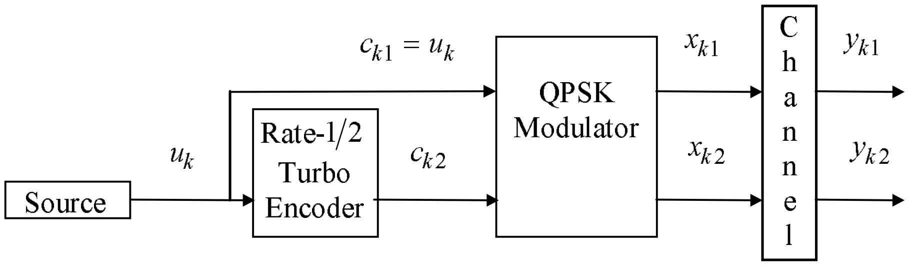

Figure 6: Rate-$\frac{1}{2}$ turbo code encoder/modulator

Computation of $\gamma_k(x(S_i,u))$ proceeds as follows. First, from the definition in (9)

$$\begin{aligned} \gamma_k(x(S_i,u)) &= P(y_k \,|\, x_k = x(S_i,u)) P(x_k = x(S_i,u)) \\ &= P(x_k = x(S_i,u)) \prod_{m=1}^{2} P(y_{km} \,|\, x_{km} = x_{km}(S_i,u)), \end{aligned}$$

since y_{k1} and y_{k2} are independent conditional on u_k. But, $u_k = u$ and $s_{k-1} = S_i$ if and only if

$$\begin{aligned} x_k(S_i,u) &= (2u-1, 2c_{k2}(S_i,u)-1) \\ &= x(S_i,u), \end{aligned}$$

and $P(u_k = u, s_{k-1} = S_i) = P(u_k = u)$, by the uniqueness of the successor of state S_i. This means that

$$P(x_k = x(S_i,u)) = P(u_k = u),$$

and hence, that

$$\gamma_k(x(S_i,u)) = P(u_k = u) \prod_{m=1}^{2} P(y_{km} \,|\, x_{km} = x_{km}(S_i,u)). \tag{18}$$

Now, from (17a) and (17b) and the assumptions on i_k and q_k, and suppressing the actual value of x_{km}, the conditional probabilities in (18) are given by

$$P(y_{km} \,|\, x_{km}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2\sigma^2}[y_{km} - x_{km}]^2\right\}. \qquad (19)$$

After substitution of (19) into (18), we get that

$$\gamma_k(x(S_i, u)) = \frac{1}{2\pi\sigma^2} P(u_k = u) \exp\left\{-\frac{D^2(y_k, x_k)}{2\sigma^2}\right\}, \qquad (20)$$

where

$$D(y_k, x_k) \triangleq \left[(y_{k1} - x_{k1})^2 + (y_{k2} - x_{k2})^2\right]^{1/2}$$

is the Euclidean distance between y_k and x_k. Since $\gamma_k(x(S_i, u))$ always depends on the output from the soft demodulator, then $\gamma_k(x(S_i, u))$ will in general contain an exponential term which is a function of $-D^2(y_k, x_k)$, as in this example.

2.1.5 Steps of the algorithm

The BCJR algorithm can now be stated as follows:

1. Initialize α_0 and β_n according to (12) and (14), respectively.

2. As each y_k is received, $\gamma_k(x)$ is computed according to (9) from $P(y_k \,|\, x_k)$ supplied by the soft demodulator and from the assumed *a priori* probabilities of the channel symbols; the soft decoder recursively computes the probabilities α_k according to (11); and the values obtained for γ_k and α_k are stored.

3. Once the entire sequence $\boldsymbol{y}$ is received, the soft decoder recursively computes the probabilities β_k according to (13) and uses them and the stored values of γ_k and α_k to compute $\sigma_k(S_i, u)$ according to (8).

Like the optimum version of the Viterbi algorithm, the original BCJR algorithm requires the complete received sequence, $\boldsymbol{y}$; however, the Viterbi algorithm requires only a forward recursion through the trellis. For this reason the storage and computational requirements of the BCJR algorithm are significantly greater than for the Viterbi algorithm.

To relieve the storage and computational burden of the original BCJR algorithm, Benedetto *et al.* [9] developed *suboptimal* modifications of the algorithm which operate sequentially on portions of the received sequence much like the common variation of the Viterbi algorithm. The first modification, called the sliding window BCJR (SW-BCJR) algorithm, operates on a fixed memory span and forces decisions with a given delay [9, section IV]. The authors provide two versions of the SW-BCJR algorithm. The second modification, called the logarithmic BCJR (Log-BCJR), converts the algorithm from multiplicative to additive form by means of a logarithmic transform [9, section V]. Two versions of this modified BCJR algorithm are also developed, with the second version using a simplifying approximation of the logarithm of the sum of exponentials. Finally, the authors mention without derivation the possibility of applying sliding windows to the Log-BCJR algorithm [9, section V]. Assuming the use of techniques similar to these, Viterbi [13] claims that the modified BCJR algorithm has a computational complexity of no more than four times that of a conventional decoder for the same code, with moderate memory requirements.

2.2 Optimal Bit Decision Decoding

We describe turbo code decoding in the context of the parallel concatenated turbo code above (Fig. 1), which is composed of: (i) a source, producing a sequence of information bits,[4] $\boldsymbol{u} = (u_1, \ldots, u_k, \ldots, u_n)$; (ii) an interleaver, producing $\widetilde{\boldsymbol{u}} = (\widetilde{u}_1, \ldots, \widetilde{u}_k, \ldots, \widetilde{u}_n)$, a bit-permuted version of $\boldsymbol{u}$; (iii) two parallel constituent convolutional codes, a modulator, and a channel which produce, respectively, code symbols, $\boldsymbol{c_i} = (c_{i1}, \ldots, c_{ik}, \ldots, c_{in})$, channel symbols, $\boldsymbol{x_i} = (x_{i1}, \ldots, x_{ik}, \ldots, x_{in})$, and received observations, $\boldsymbol{y_i} = (y_{i1}, \ldots, y_{ik}, \ldots, y_{in})$, for $i = 0, 1, 2$; and (iv) a demodulator/decoder which ultimately produces hard-decision estimates, $\widehat{\boldsymbol{u}} = (\widehat{u}_1, \ldots, \widehat{u}_k, \ldots, \widehat{u}_n)$, of the input information bits.

The optimum soft-decision outputs, conditional on *all* received observations, $\boldsymbol{y} = [\boldsymbol{y_0}, \boldsymbol{y_1}, \boldsymbol{y_2}]$, are the APP's of information bit u_k, which are defined as

$$P(u_k = u \mid \boldsymbol{y}), \text{ for } u = 0, 1. \tag{21}$$

These APP's can be computed as

$$P(u_k = u \mid \boldsymbol{y}) = \sum_{S_i} \sigma_k(S_i, u), \tag{22}$$

using the *a posteriori* transition probabilities, $\sigma_k(S_i, u)$, which were defined in (1) and are computed according to (8) by the BCJR algorithm. The log-likelihood ratio (LLR) of these APP's is defined as

$$\Lambda_k = \log \frac{P(u_k = 1 \mid \boldsymbol{y})}{P(u_k = 0 \mid \boldsymbol{y})}, \tag{23}$$

from which the optimal bit decision (OBD) for information bit u_k, based on *all* received observations, is determined by

$$\widehat{u}_k = \begin{cases} 1, \text{if } \Lambda_k > 0 \\ 0, \text{if } \Lambda_k < 0, \end{cases} \tag{24}$$

where $\widehat{u}_k$ is an estimate of u_k.

The LLR is, then, a real number representing a soft-decision, where the sign and magnitude of the LLR give, respectively, the hard decision and the reliability

[4]For simplicity of exposition, we consider only the binary case.

of that decision. To implement OBD decoding, the APP's of u_k are calculated from the outputs, $\sigma_k(S_i, u)$, of the BCJR algorithm using (22), and then Λ_k and $\widehat{u}_k$ are computed according to (23) and (24), respectively. However, because of the presence of the interleaver, it is very difficult to compute the APP's conditioned on *all* observations; therefore, the following suboptimal decoding algorithm was developed.

2.3 Suboptimal Turbo Decoding

The suboptimal iterative turbo decoding algorithm (TDA) consists of concatenated decoding modules connected by the same interleavers used in the encoding process. The algorithm is *suboptimal* because the APP's and LLR's computed by the decoding modules are based on *proper subsets* of the received observations rather than on *all* received observations as in the OBD criterion defined by (21), (23), and (24). The algorithm is *iterative* in that updated soft-decision LLR's are *repeatedly* passed from the output of one decoder to the input of another decoder with the hope that the decision based on proper subsets of the data will, after several iterations, agree with the OBD based on all of the data.[5]

For the purposes of this tutorial, we consider the turbo code example above (Fig. 1) where, for concreteness, the modulator is assumed to comprise three BPSK mappings defined by

$$x_{ik} = 2c_{ik} - 1, \text{for } i = 0, 1, 2, \tag{25}$$

and the channel is assumed Gaussian such that

$$y_{ik} = x_{ik} + q_i, \text{for } i = 0, 1, 2, \tag{26}$$

where q_i are independent, zero-mean Gaussian random variables with common variance σ^2. In this case the TDA consists of two decoding modules, one module for each constituent encoder. The computations performed in each module are described in section 2.3.1, and the linkage of the modules into an iterative feedback decoder is described in section 2.3.2.

2.3.1 Decoding Modules

In the method proposed by Berrou *et al.* [1] and expanded upon by Schlegel [8, section 8.3], the two decoding modules compute separate estimates of u_k, based on observations received from each encoder (Fig. 1). The first decoding module computes the quantity

$$\Lambda_k(C_1) = \log \frac{P(u_k = 1 \mid \boldsymbol{y_0}, \boldsymbol{y_1})}{P(u_k = 0 \mid \boldsymbol{y_0}, \boldsymbol{y_1})}, \tag{27}$$

[5] In situations encountered in many applications this hope is well-founded; however, see [11] for a counterexample in which the TDA neither agrees with the OBD nor, in fact, converges to any decision.

the LLR for u_k, conditional on the noise corrupted information bit sequence, $\boldsymbol{y_0}$, and the noise corrupted parity bit sequence, $\boldsymbol{y_1}$, of the first encoder; while the second decoding module computes the quantity

$$\Lambda_k(C_2) = \log \frac{P(u_k = 1 \,|\, \widetilde{\boldsymbol{y_0}}, \boldsymbol{y_2})}{P(u_k = 0 \,|\, \widetilde{\boldsymbol{y_0}}, \boldsymbol{y_2})}, \tag{28}$$

the LLR for u_k, conditional on the interleaved version of the noise corrupted information bit sequence, $\widetilde{\boldsymbol{y_0}}$, and the noise corrupted parity bit sequence, $\boldsymbol{y_2}$, of the second encoder, thus generating *separate* estimates, $\widehat{u}_k(C_1)$ and $\widehat{u}_k(C_2)$, of u_k. Specifically, the first decoding module computes the APP's, $P(u_k = u \,|\, \boldsymbol{y_0}, \boldsymbol{y_1})$, for $u = 0, 1$, as in (22) and (1) with $\boldsymbol{y}$ replaced by $[\boldsymbol{y_0}, \boldsymbol{y_1}]$, that is, by applying the BCJR algorithm to the trellis and received observations associated with the first encoder. Then, $\Lambda_k(C_1)$ is computed by substituting the APP's into (27), and $\widehat{u}_k(C_1)$ is obtained by comparing $\Lambda_k(C_1)$ to the zero threshold as in (24). Similarly, the second decoding module computes the APP's, $P(u_k = u \,|\, \widetilde{\boldsymbol{y_0}}, \boldsymbol{y_2})$, for $u = 0, 1$, as in (22) and (1) with $\boldsymbol{y}$ replaced by $[\widetilde{\boldsymbol{y_0}}, \boldsymbol{y_2}]$, that is, by applying the BCJR algorithm to the trellis and received observations associated with the second encoder. Finally, $\Lambda_k(C_2)$ is computed by substituting these APP's into (28), and $\widehat{u}_k(C_2)$ is obtained by comparing $\Lambda_k(C_2)$ to the zero threshold as in (24).

For computational purposes, each of the LLR's, $\Lambda_k(C_1)$ and $\Lambda_k(C_2)$, is decomposed into the sum of three terms by expanding $\gamma_k(x(S_i, u))$, as follows. First note that, according to the notational conventions of this example (Fig. 1), $y_k = (y_{0k}, y_{1k})$ and $x_k = (x_{0k}, x_{1k})$, which implies that

$$P(y_k \,|\, x_k = x(S_i, u)) = P(y_{0k}, y_{1k} \,|\, x_{0k} = x_{0k}(S_i, u), x_{1k} = x_{1k}(S_i, u)).$$

But,

$$\begin{aligned} P(y_{0k}, y_{1k} \,|\, x_{0k} = x_{0k}(S_i, u), x_{1k} = x_{1k}(S_i, u)) = {} & P(y_{0k} \,|\, x_{0k} = x_{0k}(S_i, u)) \\ & \times P(y_{1k} \,|\, x_{1k} = x_{1k}(S_i, u)), \end{aligned}$$

since y_{0k} and y_{1k} are independent, given u_k and s_{k-1}. Also,

$$\begin{aligned} P(x_k = x(S_i, u)) &= P(x_{0k} = x_{0k}(S_i, u), x_{1k} = x_{1k}(S_i, u)) \\ &= P(x_{1k} = x_{1k}(S_i, u) \,|\, x_{0k} = x_{0k}(S_i, u)) P(x_{0k} = x_{0k}(S_i, u)). \end{aligned}$$

Now, on the one hand, from (25) and since $\boldsymbol{c_0} = \boldsymbol{u}$, it follows that $x_{0k} = 2u_k - 1$; while on the other hand, since $\boldsymbol{x_0}$ does not depend on the state of the encoder, it follows that $x_{0k}(S_i, u) = 2u - 1$. This implies that $x_{0k} = x_{0k}(S_i, u)$ if and only if $u_k = u$ and, after substitution of the above equations into (9), that

$$\begin{aligned} \gamma_k(x(S_i, u)) = {} & P(y_{0k} \,|\, u_k = u) P(y_{1k} \,|\, x_{1k} = x_{1k}(S_i, u)) \\ & \times P(x_{1k} = x_{1k}(S_i, u) \,|\, x_{0k} = x_{0k}(S_i, u)) P(u_k = u). \end{aligned}$$

Substituting this last equation into (8) and using (22) and (27), leads to the three-term decomposition

$$\Lambda_k(C_1) = \Lambda_{k,apr}(C_1) + \Lambda_{k,apo}(C_1) + \Lambda_{k,e}(C_1), \tag{29}$$

where

$$\Lambda_{k,apr}(C_1) \triangleq \log \frac{P(u_k = 1)}{P(u_k = 0)}$$

is the LLR of the *a priori* probabilities of the systematic bit u_k,

$$\Lambda_{k,apo}(C_1) \triangleq \log \frac{P(y_{0k} \,|\, u_k = 1)}{P(y_{0k} \,|\, u_k = 0)}$$

is the LLR of the *a posteriori* probabilities of the systematic bit u_k, and $\Lambda_{k,e}(C_1)$ is the *extrinsic information* which is computed by subtraction. In the absence of prior information, such as on the first iteration of the TDA (see section 2.3.2), it is assumed that $P(u_k = 1) = P(u_k = 0) = 1/2$ so that $\Lambda_{k,apr}(C_1) = 0$. By the BPSK modulation assumption of (25) and the Gaussian channel assumption of (26), it follows that $\Lambda_{k,apo}(C_1) = 2y_{0k}/\sigma^2$. The extrinsic information term, which does not depend on y_{0k}, represents extra knowledge that is gained from the first decoder. By a similar analysis, it can be shown that

$$\Lambda_k(C_2) = \Lambda_{k,apr}(C_2) + \Lambda_{k,apo}(C_2) + \Lambda_{k,e}(C_2), \tag{30}$$

where

$$\Lambda_{k,apr}(C_2) \triangleq \log \frac{P(u_k = 1)}{P(u_k = 0)},$$

$$\begin{aligned} \Lambda_{k,apo}(C_2) &\triangleq \log \frac{P(\widetilde{y}_{0k} \,|\, u_k = 1)}{P(\widetilde{y}_{0k} \,|\, u_k = 0)} \\ &= 2\widetilde{y}_{0k}/\sigma^2, \end{aligned}$$

and $\Lambda_{k,e}(C_2)$ is the *extrinsic information*. This extrinsic information term is also computed by subtraction, does not depend on $\widetilde{y}_{0k}$, and represents extra knowledge gained from the second decoder.

2.3.2 Decoding Algorithm

As mentioned above, in the case of two parallel concatenated codes (Fig. 1), the TDA consists of two decoding modules which compute LLR's (27) and (28) based on the observations from each encoder. The fundamental principle of the TDA algorithm is for one decoder to provide information to the other decoder which is uncorrelated with its other inputs [1, 14]. In general terms, the TDA is a process by which the modules iteratively communicate, or feed back, their results to each other, interleaving or de-interleaving as necessary and updating their LLR's as they go, until finally $\{\widehat{u}_k(C_2)\}_{k=1}^n$ is output as the set of estimates of $\{u_k\}_{k=1}^n$. This process is depicted pictorially below (Fig. 7), where DEC1 and DEC2 represent MAP decoders corresponding to constituent encoders 1 and 2, respectively, INT represents an interleaver, and DEINT represents a de-interleaver.

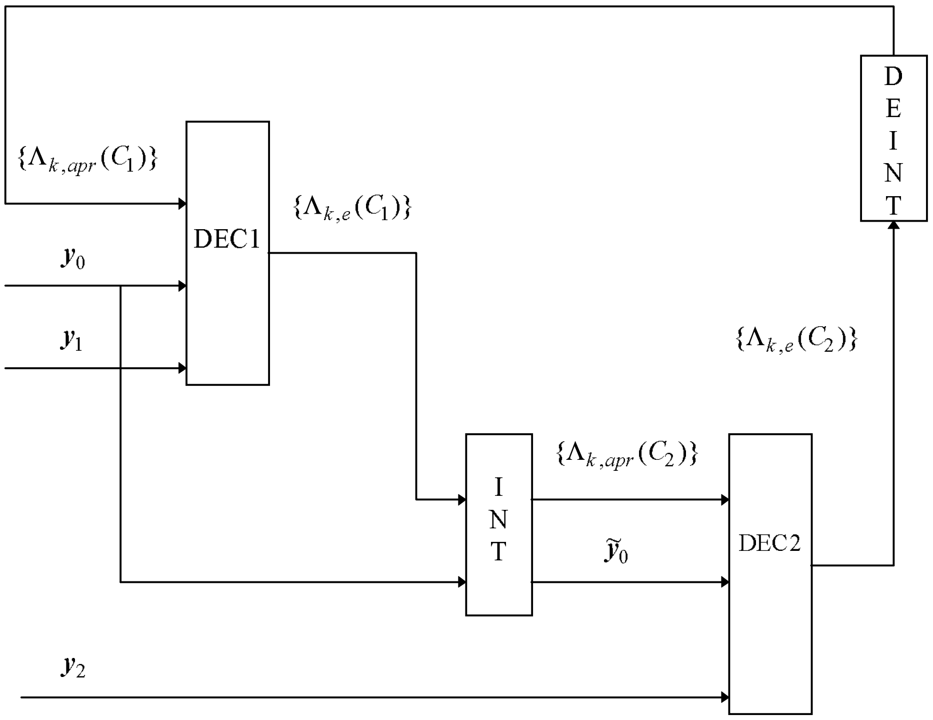

Figure 7: Iterative turbo decoder structure.

To begin the first iteration, the first decoder takes the received observations, $\boldsymbol{y_0}$ and $\boldsymbol{y_1}$, as its input and produces the LLR's, $\{\Lambda_k(C_1)\}_{k=1}^n$, via the BCJR algorithm. At this stage, there is no prior information about u_k, and it is assumed, therefore, that $u_k = 1$ and $u_k = 0$ are equally likely. So, the first decoder uses $P(u_k = u) = 1/2$ in (20) to compute the γ_k, for $k = 1, \ldots, , n$, required by step 2 of the BCJR algorithm (section 2.1.5). Then, the extrinsic information from the first decoder, $\{\Lambda_{k,e}(C_1)\}_{k=1}^n$, is computed by subtraction according to (29), where $\Lambda_{k,apr}(C_1) = 0$ and $\Lambda_{k,apo}(C_1) = 2y_{0k}/\sigma^2$. The interleaved extrinsic information, $\{\widetilde{\Lambda}_{k,e}(C_1)\}_{k=1}^n$, is fed back to the second decoder as its *a priori* information about u_k, denoted previously in (30) as $\{\Lambda_{k,apr}(C_2)\}_{k=1}^n$, and from this information together with $\widetilde{\boldsymbol{y_0}}$ and $\boldsymbol{y_2}$, the second decoder calculates the LLR's, $\{\Lambda_k(C_2)\}_{k=1}^n$, via the BCJR algorithm. The feedback of the extrinsic information, $\{\widetilde{\Lambda}_{k,e}(C_1)\}_{k=1}^{n}$, to the second decoder is accomplished by using

$$P(u_k = u) = \frac{\exp(u\widetilde{\Lambda}_{k,e}(C_1))}{1 + \exp(\widetilde{\Lambda}_{k,e}(C_1))}$$

in its calculation of the γ_k's prescribed by (20). Output of the LLR's, $\{\Lambda_k(C_2)\}_{k=1}^{n}$, completes the first iteration of the TDA.

To begin the second iteration, the extrinsic information from the second decoder, $\{\Lambda_{k,e}(C_2)\}_{k=1}^{n}$, is computed by subtraction according to (30), where $\Lambda_{k,apr}(C_2) = \widetilde{\Lambda}_{k,e}(C_1)$ and $\Lambda_{k,apo}(C_2) = 2\widetilde{y}_{0k}/\sigma^2$. The de-interleaved extrinsic information terms, $\{\overline{\Lambda}_{k,e}(C_2)\}_{k=1}^{n}$, are fed back to the first decoder as its *a priori* information about u_k, denoted previously in (29) as $\{\Lambda_{k,apr}(C_1)\}_{k=1}^{n}$, and from this information together with $\boldsymbol{y_0}$ and $\boldsymbol{y_1}$, the first decoder re-calculates the LLR's, $\{\Lambda_k(C_1)\}_{k=1}^{n}$, via the BCJR algorithm using

$$P(u_k = u) = \frac{\exp(u\overline{\Lambda}_{k,e}(C_2))}{1 + \exp(\overline{\Lambda}_{k,e}(C_2))}$$

in its calculation of the γ_k's prescribed by (20). Similarly, the second decoder re-calculates the LLR's, $\{\Lambda_k(C_2)\}_{k=1}^{n}$, and output of these terms completes the second iteration of the TDA.

In general, the benefit obtained from additional iterations is a decreased bit error probability in the estimation of the information bits, although this benefit diminishes with increasing number of iterations [1, Fig. 5]. Finally, the TDA is terminated after some number of iterations, and $\{\widehat{u}_k(C_2)\}_{k=1}^{n}$ are output after comparing the LLR's, $\{\Lambda_k(C_2)\}_{k=1}^{n}$, to the zero threshold as in (24).

2.4 Results

Plots of simulation results of bit error rate (BER) versus E_b/N_0 are provided below for two turbo encoder/modulator systems mentioned previously: a rate-$\frac{1}{3}$ turbo code with three BPSK modulators (Fig. 1) and a rate-$\frac{1}{2}$ turbo code with a QPSK modulator (Fig. 6). Simulation results for the rate-$\frac{1}{3}$ turbo encoder with BPSK modulation are plotted together with theoretical results for uncoded BPSK (Fig. 8). The turbo code comprises two 16-state RSC encoders and a moderate size 1024-bit interleaver, with results shown for 1, 3, 6, and 18 decoding iterations. Our results, showing coding gain of 7.4 dB relative to uncoded BPSK (BER $= 10^{-4}$, iterations $= 6$), are comparable to those obtained by Barbulescu [15, Fig. 3.38], and as mentioned in section 2.3.2, they show the diminishing rate of return of lower BER as the number of decoding iterations increases. BER performance can be improved with larger interleavers but at the cost of increased delay.

Simulation results for the rate-$\frac{1}{2}$ turbo encoder with QPSK modulation are plotted together with theoretical results for uncoded QPSK (Fig. 9). The turbo code comprises the same two 16-state RSC encoders as in the rate-$\frac{1}{3}$ case above and the same 1024-bit interleaver, with results shown for 1, 3, 6, and 18

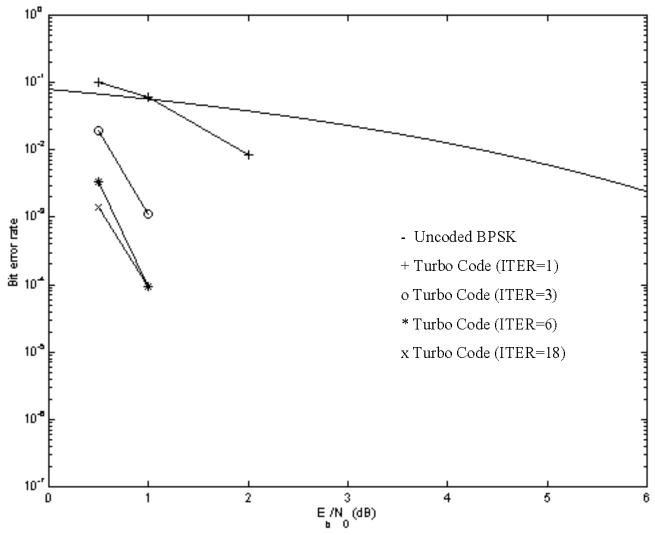

Figure 8: Bit error rate performance of rate-$\frac{1}{3}$ turbo code with BPSK modulation.

decoding iterations. Our rate-$\frac{1}{2}$ results, showing coding gain of 6.4 dB relative to uncoded QPSK (BER $= 10^{-4}$, iterations $= 6$), are comparable to those obtained by Barbulescu [15, Fig. 3.33], and as expected the rate-$\frac{1}{3}$ code achieved lower gain than the rate-$\frac{1}{2}$ code. As in the rate-$\frac{1}{3}$ case above, these results show the diminishing rate of return of lower BER as the number of decoding iterations increases, and BER performance can be improved with larger interleavers at the cost of increased delay.

3 Turbo-Coded Pragmatic TCM

The work of Ungerboeck [4] on trellis-coded modulation (TCM) and of Imai and Hirakawa [16] on multilevel coding (MLC) demonstrated that the combination of higher-order modulation schemes with set partitioning and MLC can produce coding gain without increasing transmitted power or required bandwidth. TCM, which may be interpreted as a special case of MLC with two levels, is now a well-established method in digital communication capable of achieving coding gains within the 3 to 6 dB range predicted in [4] for a trellis-coded 8-PSK

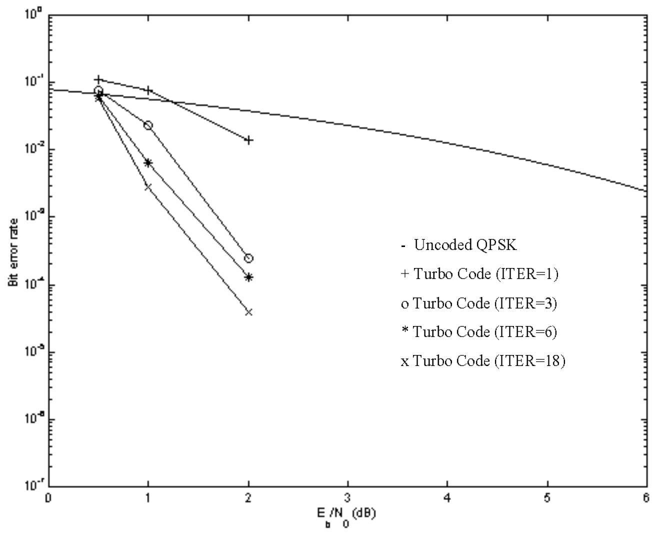

Figure 9: Bit error rate performance of rate-$\frac{1}{2}$ turbo code with QPSK modulation.

system relative to an uncoded QPSK system. Recently, Iwadate and Ikeda [17] developed a trellis-coded 8-PSK modem that can transmit High Definition Television (HDTV) programs at a bit rate of 60 Mbps over a satellite channel.

The application of turbo codes to TCM has received considerable attention in the literature; see [3,18–28] for a sample. As noted in section 1, for our application of turbo codes to TCM, we have chosen to combine turbo codes with Pragmatic TCM, creating what we call turbo-coded Pragmatic TCM (TCPTCM). Pragmatic TCM (PTCM) was described in [5] and has been implemented in hardware by Qualcomm, Inc., as the Q1900 Single Chip Viterbi/Trellis Decoder [29]. See below (Example 3) for an 8-PSK PTCM encoder/modulator, which is easily generalized to M-PSK for M a power of 2.

Example 3 *A rate-$\frac{2}{3}$ PTCM encoder/modulator (Fig. 10) receives a pair of input information bits, u_{k1} and u_{k2}; passes the most significant information bit, u_{k1}, unchanged; and encodes the least significant information bit, u_{k2}, as two bits, c_{k2} and c_{k3}, using a rate-$\frac{1}{2}$ convolutional encoder. The three encoded bits, $c_{k1} = u_{k1}$, c_{k2}, and c_{k3}, are then mapped to an 8-PSK signal constellation as specified in [5], where $c_{k1} = 0$, or 1 selects the upper or lower half-plane,* *respectively, and the combination,* $c_{k2}c_{k3}$*, selects one of four phases (in radians) by the following rule:*

$$\begin{aligned} 00 &\rightarrow 0, \\ 01 &\rightarrow \frac{\pi}{4}, \\ 11 &\rightarrow \frac{\pi}{2}, \\ 10 &\rightarrow \frac{3\pi}{4}. \end{aligned}$$

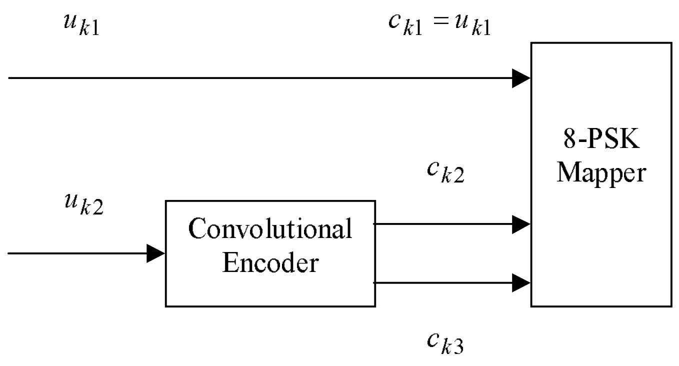

Figure 10: Rate-$\frac{2}{3}$ PTCM encoder/modulator.

3.1 Encoding and Modulation

We constructed a rate-$\frac{2}{3}$ TCPTCM encoder/modulator (Fig. 11) by analogy with the PTCM encoder/modulator in Example 3 (Fig. 10). Specifically, we made the obvious substitution of a rate-$\frac{1}{2}$ turbo encoder for the rate-$\frac{1}{2}$ convolutional encoder while keeping the same signal constellation mapping. As shown (Fig. 11) and explained previously in section 2.1.4 the rate-$\frac{1}{2}$ turbo encoder is obtained by puncturing the parity bits output from a rate-$\frac{1}{3}$ turbo encoder comprising two identical RSC codes which are concatenated in parallel and separated by an interleaver, denoted as I in the figure.

3.2 Decoding

We decode our TCPTCM code by analogy with the PTCM decoding procedure employed by the Q1900 Viterbi/Trellis Decoder as described in [29]. Briefly, decoding of rate-$\frac{2}{3}$ TCPTCM encoded data proceeds as follows:

1. Each in-phase and quadrature signal pair, $(\widehat{x}^i_k, \widehat{x}^q_k)$, received from the demodulator is input to a phase estimator which calculates a phase estimate,

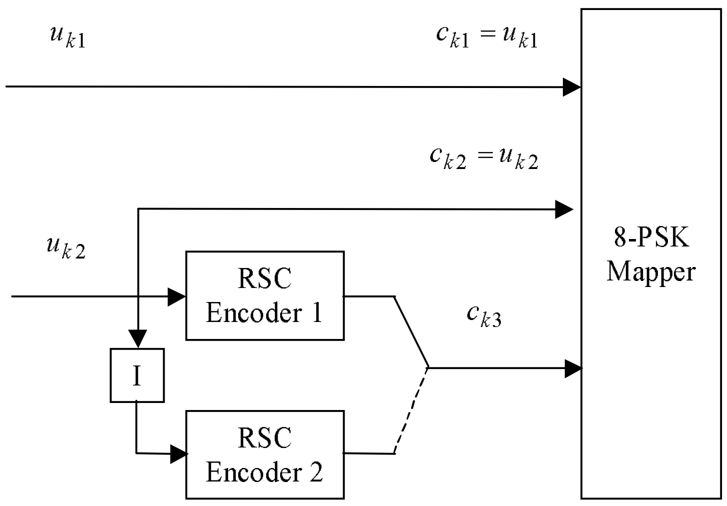

Figure 11: Rate-$\frac{2}{3}$ TCPTCM encoder/modulator.

$\widehat{\theta}_k$, providing *preliminary estimates*, $\widehat{c}_{k2}$ and $\widehat{c}_{k3}$, of c_{k2} and c_{k3}, respectively.

2. Sequences of pairs, $\widehat{c}_{k2}$ and $\widehat{c}_{k3}$, are *decoded* using a rate-$\frac{1}{2}$ turbo code decoder, providing *estimates*, $\widehat{u}_{k2}$, of u_{k2}.

3. The values of $\widehat{u}_{k2}$ are then *encoded* by the rate-$\frac{1}{2}$ turbo code encoder, providing improved *final estimates*, $\widehat{c}^*_{k2}$ and $\widehat{c}^*_{k3}$, of c_{k2} and c_{k3}, respectively.

4. Each value of $\widehat{\theta}_k$ is used again to determine whether the phase represented by the pair, $\widehat{c}^*_{k2}\widehat{c}^*_{k3}$, is in the upper or lower half-plane, thus determining an *estimate*, $\widehat{u}_{k1}$, of u_{k1}.

3.3 Results

We compared the performance of TCPTCM with that of PTCM in our own simulations and with the published simulation results of a turbo-coded TCM method of Benedetto *et al.* [19] called parallel concatenated trellis coded modulation (PCTCM). The results are plotted (Fig. 12) for comparable 2 bits/s/Hz systems: (1) uncoded QPSK; (2) the rate-$\frac{2}{3}$ PTCM system above (Example 3) with a single 64-state convolutional encoder; (3) the rate-$\frac{2}{3}$ TCPTCM system (Fig. 11) with two 16-state RSC encoders, a 1024-bit interleaver, and five decoding iterations; (4) the rate-$\frac{2}{3}$ TCPTCM system (Fig. 11) with two 16-state RSC encoders, an 8192-bit interleaver, and five decoding iterations; (5) and the rate-$\frac{2}{3}$ PCTCM system of Benedetto *et al.* [19] with two 16-state RSC encoders, four 4096-bit interleavers, and eight decoding iterations.

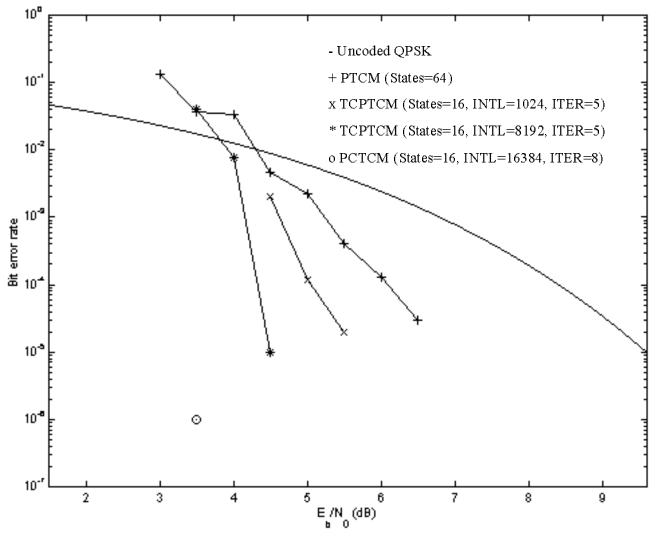

Figure 12: Bit error rate performance comparison between PTCM, TCPTCM, and PCTCM.

At a BER of 10^{-5} (Fig. 12) the PTCM simulation demonstrated coding gain of approximately 3 dB relative to the 9.6 dB attained theoretically by uncoded QPSK in AWGN; while the two TCPTCM simulations, with computational complexity comparable to the PTCM system but greater delay, achieved increased gains of 4 and 5 dB depending on interleaver length. Extrapolating the TCPTCM curve with 8192-bit interleaver to a BER of 10^{-6}, it appears that this TCPTCM system is only about 1 dB away from the 3.5 dB performance point recorded for the PCTCM system, which is itself within 1.7 dB of Shannon's limit for 2 bits/s/Hz systems.

3.4 Discussion

It is encouraging that the TCPTCM system, with simpler design comlexity and shorter interleaver, achieved performance nearly comparable to the PCTCM system. These results provide an example of the trade-offs between the benefits of very large coding gain achieved at reasonable complexity and the costs of increased delay related to interleaver length. The plots (Fig. 12) and the table below, of delay as a function of interleaver length, *INTL*, and bit rate, *R*, can be used to provide a preliminary analysis of these trade-offs.

	INTL (bits)		
R (bits/sec)	1024	8192	16384
2400	0.427	3.413	6.827
9600	0.107	0.853	1.707
12000	0.085	0.683	1.365
32000	0.032	0.256	0.512
64000	0.016	0.128	0.256

Table 2: Delay ($INTL \div R$) for selected values of *INTL* and *R*.

For further research we note that in the TCPTCM system, only one of three bits entering the modulator is encoded (Fig. 11) compared to two of three bits in the PTCM system (Fig. 10), suggesting that an improved signal mapping scheme might be developed for TCPTCM. Also, as noted above, the TCPTCM system represents a simple modification of PTCM, which has already been implemented in hardware; hence, it seems likely that, with an efficient hardware implementation of turbo encoding and decoding, a hardware implementation of the TCPTCM system itself could be developed.

4 Acknowledgment

We gratefully acknowledge the support of the Independent Research Program of the Space and Naval Warfare Systems Center, San Diego (SPAWARSYS-SD).

5 References

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima , "Near Shannon limit error-correcting codes: turbo-codes (1)," in *Proceedings of the IEEE International Conference on Communications,* Geneva, Switzerland, 23–26 May 1993, pp. 1064–1070.

[2] G. D. Forney, Jr., *Concatenated Codes.* Cambridge, MA: MIT Press, 1966.

[3] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Turbo-codes: principles and applications," Engineering 819.267 Extension Class, Univ. of Calif., Los Angeles, 23–25 Apr. 1997.

[4] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Transactions on Information Theory*, vol. IT-28, no. 1, pp. 55–67, Jan. 1982.

[5] A. J. Viterbi, J. K. Wolf, E. Zehavi, and R. Padovani, "A pragmatic approach to trellis-coded modulation," *IEEE Communications Magazine*, vol. 27, no. 7, pp. 11–19, Jul. 1989.

[6] G. D. Forney, Jr., "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[7] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. IT-20, no.2, pp. 284–287, Mar. 1974.

[8] C. Schlegel, *Trellis Coding.* New York: IEEE Press, 1997.

[9] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," *Jet Propulsion Laboratory TDA Progress Report*, vol. 42-124, pp. 63–87, 15 Feb. 1996.

[10] G. Caire, G. Taricco, and E. Biglieri, "On the convergence of the iterated decoding algorithm," in *Proceedings of the IEEE International Symposium on Information Theory*, Whistler, BC, Canada, Sep. 1995, p. 472.

[11] R. J. McEliece, E. R. Rodemich, and J.-F. Cheng, "The turbo decision algorithm," in *Proceedings of the 33rd Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Oct. 1995, pp. 366–379.

[12] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *European Transactions on Telecommunications*, vol. 6, no. 5, pp. 513–525, Sep.–Oct. 1995.

[13] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, no.2, pp. 260–264, Feb. 1998.

[14] B. Sklar, "A primer on turbo code concepts," *IEEE Communications Magazine,* vol. 35, no.12, pp. 94–102, Dec. 1997.

[15] S. A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, Univ. South Australia, 1996.

[16] H. Imai and S. Hirakawa, "A new multilevel coding method using error-correcting codes," *IEEE Transactions on Information Theory*, vol. IT-23, no. 3, pp. 371–377, May 1977.

[17] Y. Iwadate and T. Ikeda, "Development of a trellis-coded 8PSK modem and its transmission performance," *IEICE Transactions on Communications*, vol. E77-B, no. 12, pp. 1468–1473, Dec. 1994.

[18] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Bandwidth efficient parallel concatenated coding schemes," *Electronics Letters*, vol. 31, no. 24, pp. 2067–2069, 23 Nov. 1995.

[19] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Parallel concatenated trellis coded modulation," in *Proceedings of the IEEE International Conference on Communications,* Dallas, TX, 23–27 Jun. 1996, pp. 974–978.

[20] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenated trellis coded modulation with iterative decoding," in *Proceedings of the IEEE International Symposium on Information Theory,* Ulm, Germany, 29 Jun.–4 Jul. 1997, p. 8.

[21] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenated trellis coded modulation with iterative decoding: design and performance," in *Proceedings of the IEEE Global Telecommunications Conference*, Phoenix, AZ, Nov. 1997, pp. 38–43.

[22] D. Divsalar and F. Pollara, "Multiple turbo codes," in *Proceedings of the IEEE Military Communications Conference,* San Diego, CA, 5–8 Nov. 1995, pp. 279–285.

[23] D. Divsalar and F. Pollara, "On the design of turbo codes," *Jet Propulsion Laboratory TDA Progress Report*, vol. 42-123, pp. 99–121, 15 Nov. 1995.

[24] D. Divsalar and F. Pollara, "Turbo trellis coded modulation with iterative decoding for mobile satellite communications," in *Proceedings of the International Mobile Satellite Conference*, Pasadena, CA, Jun. 1997, pp. 333–342.

[25] S. Le Goff, A. Glavieux, and C. Berrou, "Turbo-codes and high spectral efficiency modulation," in *Proceedings of the IEEE International Conference on Communications,* New Orleans, LA, 1–5 May 1994, pp. 645–649.

[26] P. Robertson and T. Wörz, "A novel bandwidth efficient coding scheme employing turbo codes," in *Proceedings of the IEEE International Conference on Communications,* Dallas, TX, 23–27 Jun. 1996, pp. 962–967.

[27] P. Robertson and T. Wörz, "Bandwidth-efficient turbo trellis-coded modulation using punctured component codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, no.2, pp. 206–218, Feb. 1998.

[28] U. Wachsmann and J. Huber, "Power and bandwidth efficient digital communication using turbo codes in multilevel codes," *European Transactions on Telecommunications*, vol. 6, no. 5, pp. 557–567, Sep.–Oct. 1995.

[29] Qualcomm, Inc., "Q1900 Viterbi/trellis decoder," *Foward Error Correction Products Data Book*, vol. 80-24128-1, pp. 1-1–1-77, Feb. 1997.